MAMMOTH and Rattlesnake Workshop

Reactor physics tools for advanced multiphysics simulations

Table of Contents

MAMMOTH and Rattlesnake Overview

www.inl.gov

Idaho National Laboratory

MAMMOTH and Rattlesnake Team

- Mark DeHart
  - `Mark.DeHart@inl.gov`
- Yaqi Wang (Developer)
  - `Yaqi.Wang@inl.gov`
- Frederick Gleicher (Developer/Analyst)
  - `Frederick.Gleicher@inl.gov`
- Javier Ortensi (Developer/Analyst)
  - `Javier.Ortensi@inl.gov`
- Sebastian Schunert (Developer)
  - `Sebastian.Schunert@inl.gov`
- Benjamin Baker (Analyst)
  - `Benjamin.Baker@inl.gov`

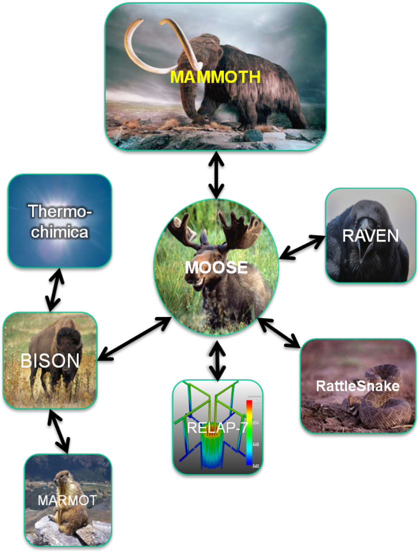
What is MAMMOTH?

- a MOOSE-based reactor physics application
- seeks to perform large scale integrated reactor multi-physics calculations on matching and non-matching geometric domains,
- built on MOOSE and inherits all the MOOSE finite element functionality,
- enables a variety of multi-physics coupling schemes (loose, tight and strong)
- able to leverage different MOOSE applications to perform large scale reactor simulations,
- being developed as a general and flexible environment to perform reactor physics analysis to support the following engineering applications:
  - core safety (ATR, TREAT, etc.)
  - transient and experiment design (TREAT),
  - core management (depletion and optimization),
  - fuel performance,
  - advanced reactor concepts (ARC) support,
  - structural Integrity,
  - spent fuel criticality and decay heat analysis

What's inside MAMMOTH?

The MAMMOTH executable is linked to the following MOOSE applications:

- Rattlesnake: radiation transport
- Yak *: radiation transport module
- BISON : fuel performance
- Relap-7 : thermal hydraulics
- Raven : reactor system and safety analysis
- Marmot : micro-structure evolution
- Thermo-chimica : chemistry



What is Rattlesnake?

- a MOOSE-based high-fidelity radiation transport application
- solves the linearized Boltzmann transport equation
- solvers and spatial discretizations:
  - CFEM: diffusion, self-adjoint angular flux (SAAF), least squares (LS)
  - DGFEM: diffusion, 1st order
- angular discretizations:
  - spherical harmonics (P_N)
  - discrete ordinates (S_N)
- arbitrary number of energy groups
- arbitrary anisotropic scattering order
- void treatment in the SAAF (2nd order) formulation
- the method of manufactured solutions is built into the transport framework

Current MAMMOTH and Rattlesnake Capabilities

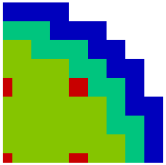
- Forward and adjoint eigenvalue problems
- Transient problems with various time integrators including the improved quasi static (IQS)
- Criticality search
- Macroscopic depletion
- Decay Heat (ANS standard based on local burnup)
- Computation of PKE parameters during transients
- Fluence determination for reactor internals and RPV (S_N solver)

Future MAMMOTH and Rattlesnake Capabilities

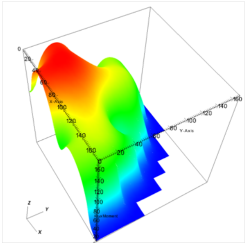
- Microscopic depletion
- Improved decay heat (based on local heating from all isotopes)
- Multi-scale capabilities
- Customizable nuclide inventory
- Gamma transport
- Integrated cross section generation
- Multi-cycle and equilibrium cycle analysis (reshuffling)

LWR Analysis - 2-D Core Assembly-Wise Eigenvalue

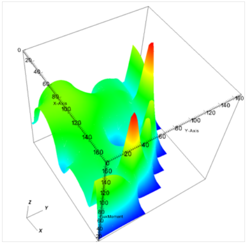
- IAEA-2D benchmark
- Standard PWR reactor
- 2 energy groups
- Rattlesnake model:
  - cross sections from the benchmark
  - assembly homogenized
  - 241 elements, quadratic shape functions
  - run-time is $<$ 1 second
- Eigenvalue 1.029696



- Red: fuel assembly with control rods
- Green: fuel
- Cyon: fuel
- Blue: reflector



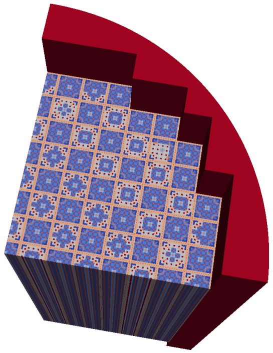
Fast flux



Thermal flux

LWR Analysis - 3-D Core Pin-Cell-Wise Eigenvalue

- Standard PWR reactor
- Start-up core (Cycle 1)
- Rattlesnake model
  - cross sections from DRAGON5
  - 3-D full core mesh
  - pin-cell homogenized
  - 8 energy groups
  - SPH corrected CFEM diffusion
  - 68 million DoFs
  - 35 minutes on 480 CPUs
- Eigenvalues
  - MIT - 0.99920 (OpenMC)
  - INL - 0.99994



-2.44%	-0.50%	-2.70%	1.01%	-0.15%	1.74%	-2.53%
-0.50%	-1.50%	0.50%	-2.42%	3.92%	-1.02%	-2.37%
	0.05%	-0.45%	-0.11%	-0.54%	3.62%	-2.13%
		1.21%		5.98%		-1.25%
			1.06%	2.15%	-1.26%	
				0.13%	-1.29%	

Rattlesnake
RMS = 2.08%
Max = 5.98%

INL error in detector fission rates

1.58%	2.41%	-0.34%	2.01%	-0.30%	-0.20%	-3.27%
2.96%	1.52%	2.42%	-1.19%	3.49%	-2.03%	-3.23%
	2.25%	1.24%	0.47%	-1.33%	1.29%	-3.22%
		1.69%		4.19%		-2.10%
			-0.50%	-0.28%	-2.74%	
				-1.45%	-2.27%	

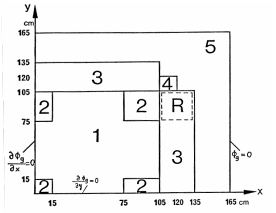
openMC
RMS = 2.14%
Max = 4.19%

MIT error in detector fission rates

LWR Analysis - Transient

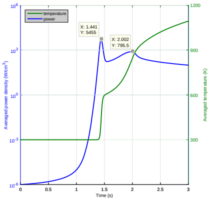
- LRA-2D benchmark (14-A1)
- BWR model
- super prompt-critical transient
- 2 energy groups
- 2 delayed neutron groups
- diffusion
- adiabatic heat-up
- assembly homogenized cross sections with Doppler feedback

Initial k-effective	0.996368
k-effective with the control rod completely out	1.015445
The first averaged peak power (W/cm^3)	5455.46
The occurrence of the first peak (s)	1.44112
The second normalized peak power (W/cm^3)	795.498
The occurrence of the second peak (s)	2.00164
Averaged power density at 0.4s (W/cm^3)	1.38437×10^{-6}
Averaged power density at 0.8s (W/cm^3)	3.07545×10^{-6}
Averaged power density at 1.2s (W/cm^3)	6.67668×10^{-6}
Averaged power density at 1.4s (W/cm^3)	432.518
Averaged power density at 2.0s (W/cm^3)	795.126
Averaged power density at 3.0s (W/cm^3)	98.4276
Averaged fuel temperature at 3.0s (K)	1094.15

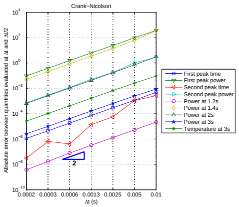


y
cm
165
135
120
105
75
15
5
3
4
2
R
1
3
2
2
$\frac{\partial \phi_g}{\partial x}=0$
$\frac{\partial \phi_g}{\partial y}=0$
$\phi_g=0$
15
75
105
120
135
165 cm
x

Geometry



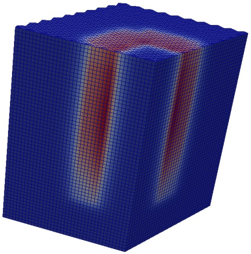
Transient



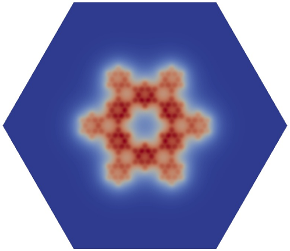
Crank-Nicholson convergence

HTR Analysis

- OECD/NEA MHTGR-350 Benchmark
- Rattlesnake model
  - cross sections from DRAGON5
  - 3-D 1/3rd core mesh
  - block homogenized
  - Triangular mesh
  - 26 energy groups
  - 26 million DoFs
- Eigenvalues
  - Mean value - 1.06743 $\pm$ 92 pcm
  - Rattlesnake - 1.06692

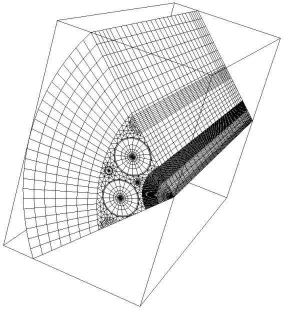


- VHTRC Benchmark
- Rattlesnake model
  - cross sections HELIOS/SERPENT
  - 3-D full core mesh
  - 1/3rd block homog. and pin resolved
  - 26 energy groups
  - 0.13 and 40 million DoFs
- Eigenvalues
  - Benchmark - 1.01100
  - Rattlesnake - 1.02036 (ENDF/B-VII.r0)



Test Reactor Analysis - ATR

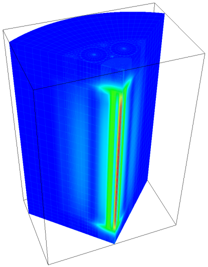
- The mesh is 3D with two lobes of ATR: (The entire ATR core contains 40 lobes.)



There are 81,004 elements of mixed types.

- The 12-group cross sections with P5 scattering was obtained from Serpent.

- The 12-th group flux:



P5 and no refinement

Sn order	k-eff
6	1.1746762
8	1.1745569
12	**1.1745146**

- k-effective results:

S8 and no refinement

Scattering order	k-eff
0	1.2772911
1	1.1719575
2	1.1746945
3	**1.1745509**
4	1.1745562
5	1.1745569

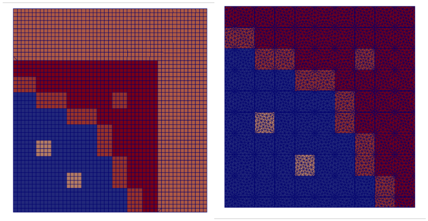
Diffusion and no refinement

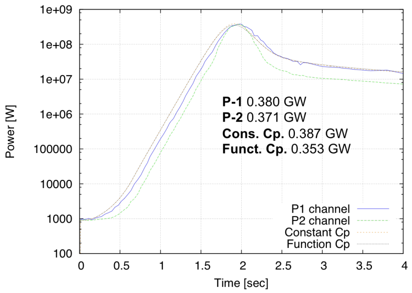
Order	Refine	k-eff
1	0	1.2471145
1	1	1.2476926
1	2	1.2486009
2	0	1.2481176
2	1	1.2489132

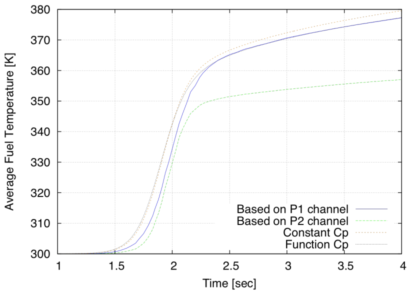
- S12-P3 is a good combination. Mesh needs to be refined. We can use AMR to generate a better mesh.

Test Reactor Analysis - TREAT

- Rattlesnake model
  - cross sections from SERPENT
  - 3-D full core mesh
  - block homog. or explicit channels
  - mixed quad and wedge elements
  - 11 energy groups
  - million DoFs
- Minimum Critical Eigenvalue
  - Rattlesnake - 1.00575





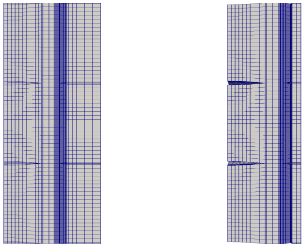


Pin coupling

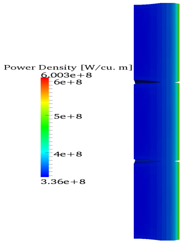
- Rattlesnake model
  - cross sections from SERPENT
  - 3-D full meshed pellets
  - void gaps and water region explicitly meshed
  - 20 radial regions to capture plutonium rim effect
  - depleted at a constant power level for a year
  - explicit dishing and chamfering
- BISON model
  - Fuel and cladding
  - UO_2 thermo-material properties internal to BISON

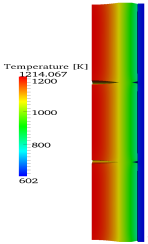
Parameter	Value
Fuel Pellet Radius	4.09575 mm
Fuel Pin Initial Temperature	600 (K)
Outer Cladding Thickness	0.5715 (mm)
Initial Gap Thickness	0.08255 (mm)
UO_2	4.45 wt %
Cladding Initial Tempareture	600 (K)
Water Temperature	585 (K)
Pitch	12.5984 (mm)
Linear Heat Rate	19.2 Wm

Rattlesnake and BISON mesh



Power Density and Fuel Temperature profile at 350 EFPF.





Getting Started

www.inl.gov

Idaho National Laboratory

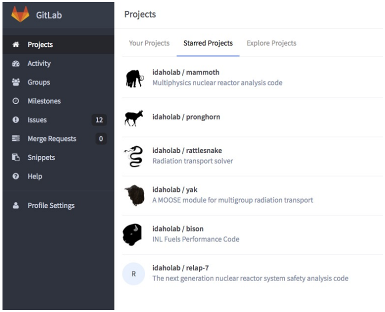
Access to MAMMOTH

- MAMMOTH, Rattlesnake, Yak, Bison, Relap-7, and Pronghorn are hosted on an INL internal gitlab server.
- Moose is hosted on github and is publicly available.
- You must meet requirements to access these codes (no exceptions):
  - You must work for INL or your organization must have a license agreement with INL.
  - INL HPC account: If you work for INL, you can request an account. If you do not work for INL, you have to have a sponsor inside INL.
- Rattlesnake, Yak, Bison, Relap-7, and Pronghorn are available through MAMMOTH if access is available.
- Process to obtain access to MAMMOTH:
  1. (Foreign nationals only) All apps that your request have to be added to your security plan.
  2. You have to apply for an HPC account.
  3. The developers have to add your as a member to access the project (the ”animal”) on gitlab. You need to be a member of each application that you will use!

Getting Started-I

- If you are within the INL domain you can access the INL gitlab server here: https://hpcgitlab.inl.gov
- If you are not connected to the INL domain:
  - Set up a SOCKS proxy with localhost set to 5555.
  - Connect to INL HPC.
    ssh -D 5555 username@hpclogin.inl.gov
- Login with HPC username and password.

- In your projects page you will see a list of ”animals” that you can access.
- If you cannot find the idaholab/animal you are looking for in the list, you do not have access granted.
- Select idaholab/mammoth.



idaholab / mammoth
Multiphysics nuclear reactor analysis code

idaholab / pronghorn

idaholab / rattlesnake
Radiation transport solver

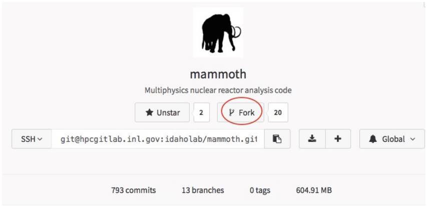
idaholab / yak
A MOOSE module for multigroup radiation transport

idaholab / bison
INL Fuels Performance Code

R idaholab / relap-7
The next generation nuclear reactor system safety analysis code

Getting Started-II

- A ”fork” is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.
- From the application menu click on the ”Fork” button.



```
git@hpcgitlab.inl.gov:idaholab/mammoth.git
```

- The new page should indicate ”Forked from idaholab.”
- Copy the SSH URL.
- More information here: `https://mooseframework.org/static/media/uploads/docs/moose_gitlab.pdf`

Getting Started-III

1. For local installations follow the link in the MOOSE wiki page of your platform to install the redistributable packages.
2. Change directory into projects and use git to clone the code into ~/projects by pasting the SSH URL.

```
>cd ~/projects
>git clone git@hpcgitlab.inl.gov:USERNAME/mammoth.git
```

3. Initialize the submodules that you will need and build libMesh.

```
>cd ~/projects/mammoth
>git submodule update --init -- ./moose/
>git submodule update --init -- ./rattlesnake/
>git submodule update --init -- ./yak/
 ... initialize other submodules as necessary (i.e. bison, pronghorn, relap-7)
```

4. The submodule update might fail with the following error message ”SSL certificate problem: Invalid certificate chain”. In this case do the following:

```
>cd .git
>Open config in your text editor of choice.
>Replace "url = https://hpcgitlab.inl.gov/idaholab/mammoth.git"
  by
  "url = git@hpcgitlab.inl.gov:idaholab/mammoth.git"
```

Getting Started-IV

5. Build libMesh.

```
>cd ~/projects/mammoth/moose/scripts
>./update_and_rebuild_libmesh
```

6. Build MAMMOTH.

```
>cd ~/projects/mammoth
>make -j8 (Add -j8 to compile using 8 processors.)
```

7. Run the regression tests to verify that MAMMOTH was build correctly.

```
>./run_tests -j8
```

8. Run the regression tests for all compiled applications with:

```
>git submodule foreach 'make -j8 test'  (FIX ME SEBASTIAN)
```

9. Done.

Keeping the Applications up to Date

1. Update MAMMOTH

```
>git pull --rebase upstream devel
```

2. Update the submodules

```
>git submodule update
```

3. Compile

```
>make -j8   (Add -j8 to compile using 8 processors.)
```

If libmesh has been updated or significant changes have occurred, you might need to clean all applications before building:

```
>make cleanall
>make -j8   (Add -j8 to compile using 8 processors.)
```

Building Models

Building Models - Overview

There are three general items needed for a MAMMOTH or Rattlesnake simulation

- Mesh(s)
  - some MOOSE applications might require independent meshes
  - this will require the projection of the solution space
  - try to keep the mesh as consistent as possible to minimize interpolation errors
  - for large # of cross section regions use a material ID variable in the mesh
- Cross sections
  - these can originate from any source, but need to be converted to our internal format
  - we provide a basic converter for Serpent, that will be expanded to other codes
  - we plan to add some limited cross section generation capability in the future
- Input file(s)
  - some multiphysics problems might require separate input files for each MOOSE application

Generating a Mesh

www.inl.gov

Idaho National Laboratory

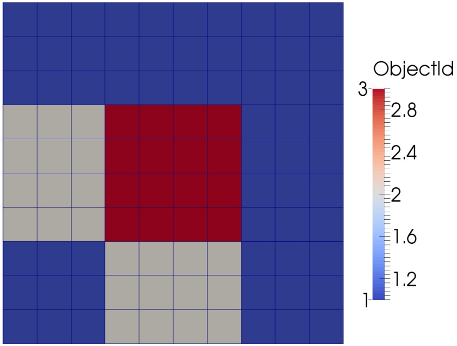
Meshing - Tools Available

- MOOSE and MAMMOTH based - INTERNAL
  - provide build-in generators for regular **Cartesian** and **Hexagonal** geometries
- INSTANT Mesh Generator - INTERNAL
  - A seperate nodal neutronics package (INSTANT) contributes to Yak.
  - Included in *rattlesnake/contrib/instant*
  - Provides a few more geometries like LWR, and PSLG (Planar Straight Line Graph). See INSTANT documentation for more details.
  - INSTANT also provides a mesh utility that analyzes the mesh for hanging nodes or separated domains.
- CUBIT - EXTERNAL
  - U.S. contractors or government facilities have access to CUBIT
  - U.S. universities can obtain university licenses
  - some of our python scripts will be included in *mammoth/contrib/mesher*
- Other EXTERNAL (MeshKit, Gmsh, etc.)

Cartesian Mesh

- Extensions of MOOSE’s GeneratedMesh
- GeneratedBIDMesh - has fixed element sides
- CartesianMesh - allows flexible element sides
- Includes a subdomain parameter, which allows to assign a block (subdomain) ID to each mesh cell.

```
[Mesh]
 type = GeneratedBIDMesh
 dim = 2
 xmin = 0
 xmax = 80
 ymin = 0
 ymax = 80
 elem_type = QUAD9
 nx = 10
 ny = 10
 subdomain='1 1 1 2 2 2 2 1 1 1
            1 1 1 2 2 2 2 1 1 1
            1 1 1 2 2 2 2 1 1 1
            2 2 2 3 3 3 3 1 1 1
            2 2 2 3 3 3 3 1 1 1
            2 2 2 3 3 3 3 1 1 1
            2 2 2 3 3 3 3 1 1 1
            1 1 1 1 1 1 1 1 1 1
            1 1 1 1 1 1 1 1 1 1
            1 1 1 1 1 1 1 1 1 1'

 uniform_refine = 0
[]
```

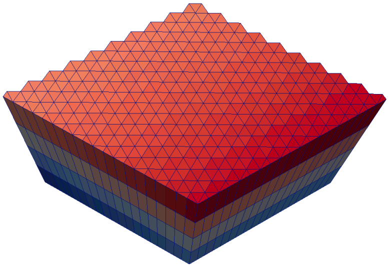


Prismatic Mesh - 1/3 Core

```
[Mesh]
 type=PrismaticReactorMeshThird
 material_map_file = m3.dat
 generated_mesh_filename = 'Hex_3.e'
 mesh_pattern = SIMPLE
 rotation = 60
 fuel_assembly_pitch = .36
 delta_z = '4'
 axial_divisions = '4'
 left   = 1
 right  = 2
 outer  = 3
 bottom = 4
 top    = 5
[]
```

```
Contents of m3.dat
75  76  77  78  79  80  81  82  83
 59  60  61  62  63  64  65  66  84
  45  46  47  48  49  50  51  67  85
   33  34  35  36  37  38  52  68  86
    23  24  25  26  27  39  53  69  87
     15  16  17  18  28  40  54  70  88
       9  10  11  19  29  41  55  71  89
        5   6  12  20  30  42  56  72  90
         3   7  13  21  31  43  57  73  91
     1   2   4   8  14  22  32  44  58  74
```

- Builds 1/3 core geometry based on material map file.
- Stack maps in material map file to assign SubdomainID to various layers.

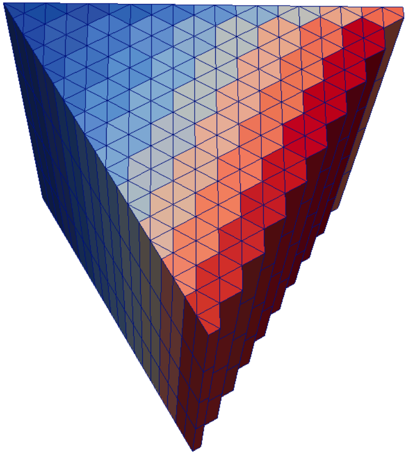


Prismatic Mesh - 1/6 Core

```
[Mesh]
 type=PrismaticReactorMeshSixth
 material_map_file = m6.dat
 generated_mesh_filename = 'Hex_6.e'
 mesh_pattern = SIMPLE
 rotation = 30
 fuel_assembly_pitch = .36
 delta_z = '4'
 axial_divisions = '4'
 left   = 1
 right  = 2
 outer  = 3
 bottom = 4
 top    = 5
[]
```

```
Contents of m6.dat
 1   2    4   8 14 22 32 44 58 74
 3   7  13 21 31 43 57 73 91
 6 12  20 30 42 56 72 90
12 20 30 42 56 72 90
19 29 41 55 71 89
28 40 54 70 88
39 53 69 87
52 68 86
67 85
84
```

- Builds 1/6 core geometry based on material map file.

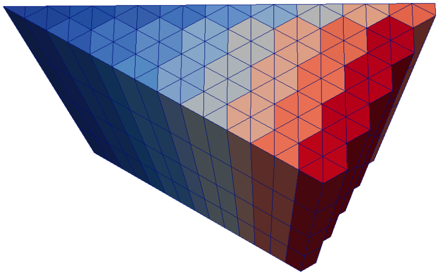


Prismatic Mesh - 1/12 Core

```
[Mesh]
 type=PrismaticReactorMeshTwelfth
 material_map_file = m12.dat
 generated_mesh_filename = 'Hex_12.e'
 mesh_pattern = SIMPLE
 rotation = 30
 fuel_assembly_pitch = .36
 delta_z = '4'
 axial_divisions = '4'
 left   = 1
 right  = 2
 outer  = 3
 bottom = 4
 top    = 5
[]
```

```
Contents of m12.dat
1 2  4  8 14 22 32 44 58 74
   7 13 21 31 43 57 73 91
      20 30 42 56 72 90
           42 56 72 90
                71 89
```

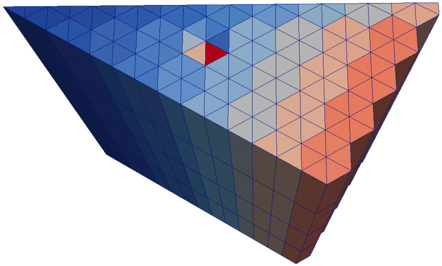
- Builds 1/12 core geometry based on material map file.



Prismatic Mesh - Burnable Poisons and Control Rods

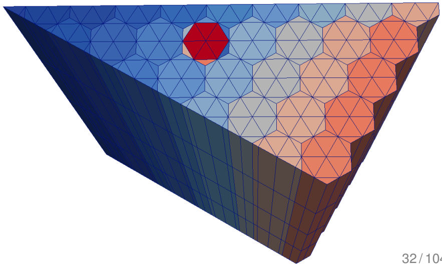
- Control Rods can be added with the following additional entries.
- Here subID 31 in m12.dat is replaced by a list of 6 new materials for CRs.

```
[Mesh]
  .
 control_rod_block= '31'
 control_rod_map= '1 2 3 4 5 6'
 []
```



- BP regions can be added with the following additional entries
- Here subID 31 in m12.dat is replaced by a list of 7 new materials for BPs.

```
[Mesh]
  .
  burnable_poison_block = '31'
  burnable_poison_map = '1 2 3 4 5 6 7'
 []
```



Meshing - Exercise ms_1

This exercise teaches the student how to build a mesh with GeneratedBIDMesh.

- use the sample input block provided $\rightarrow$
- type is GeneratedBIDMesh
- build a 2-D mesh with:
  - dimensions x [0,100] y [0,50]
  - with interval x 10 and y 5
  - element type QUAD4
- **note** that the mesh is in centimeters

```
[Mesh]
 type =
 dim =
 xmin =
 xmax =
 ymin =
 ymax =
 nx =
 ny =
 elem_type =
 subdomain='1 1 1 1 1 1 1 1 1 3
            1 1 1 1 1 1 1 1 1 3
            1 1 1 1 1 1 1 1 2 3
            1 1 1 1 1 1 1 2 2 3
            3 3 3 3 3 3 3 3 3 3'
 uniform_refine = 0
[]
```

Meshing - Checking your Mesh

- Often times convergence or solution shape problems are due to the mesh
- Using the INSTANT mesh checker

```
# to compile INSTANT and the mesh checker
~/projects/mammoth/rattlesnake>make instant -j8

# to execute the mesh checker under rattlesnake
~/projects/mammoth/rattlesnake/contrib/instant/instant_mesh_generator-opt inp.xml

# see the INSTANT manual for details on how to prepare the input file in
~/projects/mammoth/rattlesnake/contrib/instant/doc/manual.pdf
```

- Use the Debug block in MOOSE with:

```
  [Debug]
    check_boundary_coverage = 1
    print_block_volume = 1
  []
```

Data Preparation

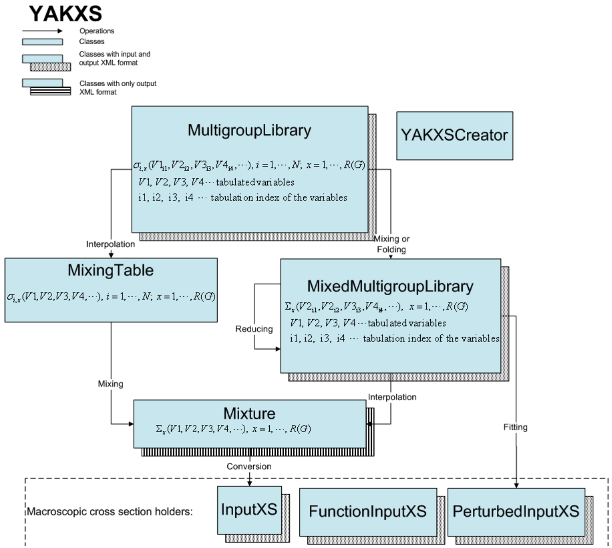
MAMMOTH

- A multi-group reactor physics database includes among other items these basic datasets:
  - neutron and gamma cross sections (cross sections, diffusion coefficients, velocities)
  - delayed neutron (delayed neutron fractions and spectrum)
  - depletion data (decay constants, branching ratios, Q values, other decay chain information)
- Present state and Future Development
  - MAMMOTH and Rattlesnake support both INSTANT and YAKXS formats (both internal INL formats) for cross sections and delayed neutron data.
  - We are working on our decay and reaction library formats.
  - Preliminary capability to convert SERPENT to YAKXS and INSTANT format is available.
  - Support planned for the following external neutron cross section formats: DRAGON-5, SERPENT (branch), OpenMC and AMPX.

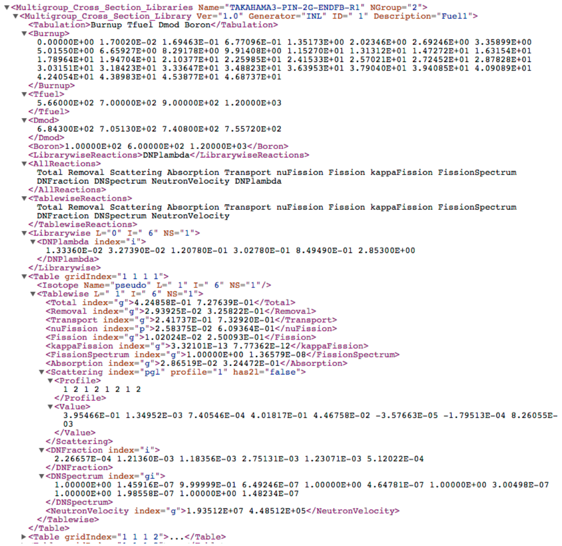
YAKXS

- YAKXS (Yak cross section library) is a modern XML (eXtendable Markup Language) multigroup cross section library format.
- Cross section processing functions, including interpolation, mixing, etc., are provided along with the library format.
- It is designed to provide a general tool for radiation transport calculations but not limited to be used only by Yak.
- The source files are located under *rattlesnake/contrib/yakxs*.
- The main source file is yakxs.C, which defines the MultigroupLibrary, MixingTable and Mixture.
- A separate document for YAKXS can be found under *rattlesnake/contrib/yakxs/doc*.

YAKXS Graphical View


YAKXS
Operations
Classes
Classes with input and output XML format
Classes with only output XML format
MultigroupLibrary
$\sigma_{i,x}(V1_{i1},V2_{i2},V3_{i3},V4_{i4},\cdots),\ i=1,\cdots,N;\ x=1,\cdots,R(G)$
$V1,\ V2,\ V3,\ V4\cdots$ tabulated variables
i1, i2, i3, i4 ⋯ tabulation index of the variables
YAKXSCreator
Interpolation
Mixing or Folding
MixingTable
$\sigma_{i,x}(V1,V2,V3,V4,\cdots),\ i=1,\cdots,N;\ x=1,\cdots,R(G)$
MixedMultigroupLibrary
$\Sigma_x(V2_{i1},V2_{i2},V3_{i3},V4_{i4},\cdots),\ x=1,\cdots,R(G)$
$V1,\ V2,\ V3,\ V4\cdots$ tabulated variables
i1, i2, i3, i4 ⋯ tabulation index of the variables
Reducing
Mixing
Interpolation
Fitting
Mixture
$\Sigma_x(V1,V2,V3,V4,\cdots),\ x=1,\cdots,R(G)$
Conversion
Macroscopic cross section holders:
InputXS
FunctionInputXS
PerturbedInputXS

YAKXS - Parametric Micro/Macroscopic Cross Sections



```
▼<Multigroup_Cross_Section_Libraries Name="TAKAHAMA3-PIN-2G-ENDFB-R1" NGroup="2">
 ▼<Multigroup_Cross_Section_Library Ver="1.0" Generator="INL" ID=" 1" Description="Fuel1">
    <Tabulation>Burnup Tfuel Dmod Boron</Tabulation>
   ▼<Burnup>
      0.00000E+00 1.70020E-02 1.69463E-01 6.77096E-01 1.35173E+00 2.02346E+00 2.69246E+00 3.35899E+00
      5.01550E+00 6.65927E+00 8.29178E+00 9.91408E+00 1.15270E+01 1.31312E+01 1.47272E+01 1.63154E+01
      1.78964E+01 1.94704E+01 2.10377E+01 2.25985E+01 2.41533E+01 2.57021E+01 2.72452E+01 2.87828E+01
      3.03151E+01 3.18423E+01 3.33647E+01 3.48823E+01 3.63953E+01 3.79040E+01 3.94085E+01 4.09089E+01
      4.24054E+01 4.38983E+01 4.53877E+01 4.68737E+01
    </Burnup>
   ▼<Tfuel>
      5.66000E+02 7.00000E+02 9.00000E+02 1.20000E+03
    </Tfuel>
   ▼<Dmod>
      6.84300E+02 7.05130E+02 7.40800E+02 7.55720E+02
    </Dmod>
    <Boron>1.00000E+02 6.00000E+02 1.20000E+03</Boron>
    <LibrarywiseReactions>DNPlambda</LibrarywiseReactions>
   ▼<AllReactions>
      Total Removal Scattering Absorption Transport nuFission Fission kappaFission FissionSpectrum
      DNFraction DNSpectrum NeutronVelocity DNPlambda
    </AllReactions>
   ▼<TablewiseReactions>
      Total Removal Scattering Absorption Transport nuFission Fission kappaFission FissionSpectrum
      DNFraction DNSpectrum NeutronVelocity
    </TablewiseReactions>
   ▼<Librarywise L="0" I=" 6" NS="1">
     ▼<DNPlambda index="i">
        1.33360E-02 3.27390E-02 1.20780E-01 3.02780E-01 8.49490E-01 2.85300E+00
      </DNPlambda>
    </Librarywise>
   ▼<Table gridIndex="1 1 1 1">
      <Isotope Name="pseudo" L=" 1" I=" 6" NS="1"/>
     ▼<Tablewise L=" 1" I=" 6" NS="1">
        <Total index="g">4.24858E-01 7.27639E-01</Total>
        <Removal index="g">2.93925E-02 3.25822E-01</Removal>
        <Transport index="g">2.41737E-01 7.32920E-01</Transport>
        <nuFission index="p">2.58375E-02 6.09364E-01</nuFission>
        <Fission index="g">1.02024E-02 2.50093E-01</Fission>
        <kappaFission index="g">3.32101E-13 7.77362E-12</kappaFission>
        <FissionSpectrum index="g">1.00000E+00 1.36579E-08</FissionSpectrum>
        <Absorption index="g">2.86519E-02 3.24472E-01</Absorption>
       ▼<Scattering index="pgl" profile="1" has2l="false">
         ▼<Profile>
            1 2 1 2 1 2 1 2
          </Profile>
         ▼<Value>
            3.95466E-01 1.34952E-03 7.40546E-04 4.01817E-01 4.46758E-02 -3.57663E-05 -1.79513E-04 8.26055E-
            03
          </Value>
        </Scattering>
       ▼<DNFraction index="i">
          2.26657E-04 1.21360E-03 1.18356E-03 2.75131E-03 1.23071E-03 5.12022E-04
        </DNFraction>
       ▼<DNSpectrum index="gi">
          1.00000E+00 1.45916E-07 9.99999E-01 6.49246E-07 1.00000E+00 4.64781E-07 1.00000E+00 3.00498E-07
          1.00000E+00 1.98558E-07 1.00000E+00 1.48234E-07
        </DNSpectrum>
        <NeutronVelocity index="g">1.93512E+07 4.48512E+05</NeutronVelocity>
      </Tablewise>
    </Table>
   ▶<Table gridIndex="1 1 2">...</Table>
```

Preparing the Input

MAMMOTH versus Rattlesnake Recap

- Rattlesnake solves the multigroup neutron transport equation
  - forward and adjoint eigenvalue
  - steady state source problem
  - time dependent with precursor equations (0-D PKE and 3-D spatial kinetics)
- Rattlesnake can also solve the radiative transfer equation (RTE) (but not covered in this traning)
- MAMMOTH controls the execution of various calculations needed in multiphysics simulations
  - perform upper level computations that require multiple solution spaces (e.g. depletion, feedback, SPH correction, optimization)
  - calls Rattlesnake to obtain flux solutions (e.g. eigenvalue calculation)
  - can call BISON, Relap-7 or Pronghorn to obtain thermal and fluid fields

The Transport Systems

Multigroup Neutron Transport Equation

The neutron transport physics can be modeled with the multigroup neutron transport equation:

$$\frac{1}{v_g}\frac{\partial}{\partial t}\Psi_g(\vec{r},\vec{\Omega},t) + \vec{\Omega}\cdot\vec{\nabla}\Psi_g + \sigma_{t,g}(\vec{r})\Psi_g = S_{ext,g}(\vec{r},\vec{\Omega},t) + \boxed{\frac{\chi_{p,g}(\vec{r})}{4\pi}\sum_{g'=1}^{G}\nu_p\sigma_{f,g'}(\vec{r})\int_{4\pi}\Psi_{g'}(\Omega')d\Omega'} +$$

$$\boxed{\frac{1}{4\pi}\sum_{d=1}^{I}\chi_{d,g}(\vec{r})\lambda_d C_d(\vec{r},t)} + \sum_{g'=1}^{G}\sigma_{s,g'}(\vec{r})\int_{4\pi}p_{g'\to g}(\vec{r},\vec{\Omega}'\cdot\vec{\Omega})\Psi_{g'}(\Omega')d\Omega', \qquad g = 1,\cdots,G.$$

- g - group index. p, d - prompt, delayed. G - number of energy groups. I - number of delayed neutron groups.
- independent variables: t - time; $\vec{r}$ - position; $\vec{\Omega}$ - streaming direction.
- Ψ_g: angular group flux in group g (the primary variable!) .
- C_d: neutron precursor concentration.
- v_g, $\sigma_{t,g}$, χ_g, ν, $\sigma_{f,g}$, $\sigma_{s,g}$: group averaged neutron speed, total cross section, fission spectrum, fission yield, fission cross section, scattering cross section.
- $p_{g'\to g}(\vec{r},\vec{\Omega}'\cdot\vec{\Omega})$: scattering kernel.
- $S_{ext,g}$: external source.
- Notes:
  - fission and fission-related delayed neutron production are special for neutron transport;
  - the equations for delayed neutron precursors are not given here;
  - boundary conditions on the incoming angular fluxes are required.
  - because of the fission production with ($\nu > 1$), neutron population of a system can be sustained without external source;
  - scattering can be treated with spherical harmonics expansion with relatively a low scattering order for neutron transport;
  - multiphysics feedback on neutron distribution are in place through those cross sections;

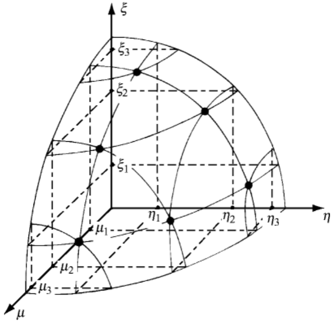
Choosing the Discretization Schemes

- The multigroup neutron transport equation must be discretized to be solved numerically.
- FEM (finite element methods) are used exclusively for *spatial* discretization; S_N (discrete ordinates method) or P_N (spherical harmonics expansion method) are used for *angular* discretization (Diffusion can be thought as a special case of P_N); Method of lines is applied to *time*, i.e. time integration schemes are independent on neutron transport and any of them can be used.
- Currently supported schemes by Rattlesnake:

Scheme	Mathematical adjoint	Neutron	Thermal radiation	Transient	Multiscale
CFEM-Diffusion	Y	Y	Y	Y	Y
DFEM-Diffusion	N	Y	N	Y	N
SAAF-CFEM-SN	Y	Y	Y	Y	Y
SAAF-CFEM-PN	N	Y	N	Y	Y
LS-CFEM-SN	N	Y	N	Y	N
LS-CFEM-PN	N	N	N	N	N
DFEM-SN	Y	Y	N	Y	N
DFEM-PN	N	N	Y	N	N

Angular Discretization Methods in Rattlesnake

- Discrete ordinates (S_N)
  - Transport equation solved along discrete streaming directions.
  - The directions in the streaming term are decoupled.
  - The computing effort is increased about linearly with the increased number of streaming directions.
  - Accuracy depends on order and type of angular quadrature.
    - Level symmetric (maintains symmetry - cheap since less unknowns)
    - Gauss-Chebyshev (allows specification of polar and azimuthal directions)
    - Bickley3-Optimized (production quadrature for 2D)
  - Prone to ray effects (or solution singularities).



- Spherical harmonics expansion (P_N)
  - Directional dependence approximated by series of spherical harmonics functions.
  - Unknowns are the directional expansion coefficient or angular moments.
  - Streaming term couples all angular moments.

$$Y_{l,m\geq 0}(\vec{\Omega}) \equiv \sqrt{\frac{(l-m)!}{(l+m)!}(2-\delta_{m,0})}P_l^m(\mu)\cos(m\theta)$$

$$Y_{l,m<0}(\vec{\Omega}) \equiv \sqrt{2\frac{(l-|m|)!}{(l+|m|)!})}P_l^{|m|}(\mu)\sin(|m|\ \theta)$$

- S_N is typically more suitable for heterogeneous calculations, while P_N is mainly for problems with the significant spatial homogenization.

FEM for Spatial Discretization in Rattlesnake

- Continuous FEM
  - stabilization scheme needs to be applied for hyperbolic system like the neutron transport equation
  - ***pros*** most of existing mesh frameworks support directly; AMG (algebraic multi-grid method) can be used.
  - ***cons*** having requirements on the mesh quality.
  - ***cons*** memory intensive because of storing the preconditioning matrix.
  - ***cons*** lack of local conservation.
  - ***cons*** high-order expansion functions are expensive.
- Discontinuous FEM
  - simple upwind scheme
  - DFEM can be solved with the transport sweep
  - ***pros*** highly memory efficient with the matrix-free scheme;
  - ***cons*** difficulties associated with transport sweep: massive parallelization, unstructured mesh, etc.
- Rattlesnake currently does not have a good transport sweeper. So for large size applications you will want to choose CFEM.

The Input Syntax of Transport Systems - Overview

```
[TransportSystems]
  particle = neutron
  equation_type = eigenvalue
  G = 8
  VacuumBoundary ='bottom top'
  ReflectingBoundary ='right left'

  [./saaf]
     scheme = SAAF-CFEM-SN
     block = '1 2'
     family = LAGRANGE
     order = FIRST
     AQtype = Gauss-Chebyshev
     NAzmthl= 2
     NPolar = 3
     NA = 2
  [../]
[]
```

- Allows users to specify:
  - transport equation to solve
  - discretization scheme
  - spatial domain where transport equation is solved
  - spatial functional space
  - angular discretization
  - angular discretization settings
  - scattering order (number of anisotropy)
  - and etc...
- Multiscale can be easily turned on by providing multiple block-restricted discretization sub-blocks.

Neutronics Materials

www.inl.gov

Idaho National Laboratory

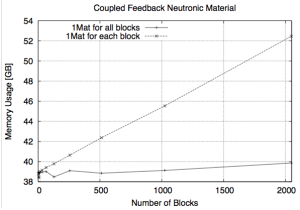
Neutronics Materials - Overview

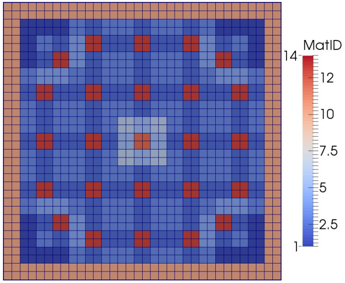
- It declares and evaluates material properties (cross sections) required by neutronics computations for both diffusion and transport, steady-state and transient.
- Types of material (Rattlesnake user manual contains full list):
  - **Constant** - specifies a constant set of macroscopic cross sections
  - **Mixed** - creates a fixed set of macroscopic cross sections by interpolating and mixing from a tabulated cross section set
  - **CoupledFeedback** - similar to Mixed, but with on-the-fly interpolation and mixing from a tabulated cross section set
  - **Function** - all macroscopic cross sections can be a function varying in space and time. Suitable for doing MMS or some complicated spatial kinetics benchmark calculations
  - **CRodded** - to model control rod movement within a block
- Volume correction of cross sections is available if the meshed volume is different from the 'real' volume.
- We will discuss two materials: 1) **Constant** and 2) **CoupledFeedback**.

Neutronics Materials - Assigning cross sections

1. Using the block number
   - simplest XS assignment
   - uses the 'block' parameter to match 1 block # to 1 XS ID in library.
   - used for a small # of XS regions.
   - can be automated using the last two digits of the block # to match library ID.

2. Using a material ID variable
   - when the number of material regions in the problem is large, the material input block can become extremely long.
   - the memory usage increases for large numbers of blocks ($>$ 100).
   - assigns cross sections based on a material ID variable written to the mesh.





Neutronics Materials - Constant

From INSTANT XML file

```
      fromFile = 1
      fileName = myXSfile.xml
      [../]
     []
```

From direct input

```
    sigma_t = '1.0 2.0'
    sigma_s = '0.99 0.0
               0.01 0.98'
    nu_sigma_f = '0.01 0.02'
    chi = '0.0 1.0'
    diffusion_coef = '1.0 2.0'
    sigma_r = '0.8 1.8'
    sigma_f = '0.01 0.02'
    kappa_sigma_f = '0.0001 0.0002'

    neutron_speed = '1e6 220000'
    decay_constant = '1.0 2.0'
    delay_fraction = '0.1 0.2'
    delay_spectrum = '0.0 1.0
                      0.1 1.0'
    [../]
   []
```

- Can input cross sections by either:
  - reading from a file
  - direct input
- In direct input we specify the values of the main cross sections: total, scattering, nu fission, fission spectrum, diffusion coefficients, removal, fission, and kappa fission.
- Note that some of these can be internally calculated - depending on calculation type (diffusion vs transport).
- Other data like neutron speed and delayed neutron data is also included in the material specification.

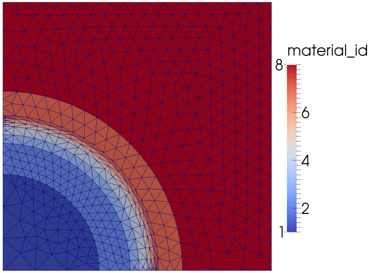
Neutronics Materials - CoupledFeedback

- ***CoupledFeedbackNeutronicsMaterials*** is used with parametric data $\Sigma(Burnup, fueltemp, modtemp, bor, ...)$.
- Tied to a BaseLibObject (userobject), which loads cross section data and interacts with the material.
- The material will use the values of MOOSE variables (burnup, fuel temperature) at each quadrature point to obtain an interpolated table.
- The MOOSE variables can either be primal or Aux, but order of execution will be important for the latter
- The material will call a mixer to generate a set of macroscopic cross sections.
- Requires the isotope names and number densities of initial isotopes. The use of isotope ”pseudo” indicates it they are macro cross sections with number density 1.0.

Neutronics Materials - The use of material IDs

- Some specialized versions of the neutronics materials allow automatic assignment of the material_id based on either:
  - last two digits of the block number
  - the value of a material_id variable in the mesh file

- This specialized neutronics materials use the same naming convention but with "MatID" inserted after the base name:
  - ConstantMatIDNeutronicsMaterial
  - MixedMatIDNeutronicsMaterial
  - CoupledFeedbackMatIDNeutronicsMaterial
- **If reading a material ID variable from the mesh file make sure that the mesh distribution is serial. Failure to do this will:**
  - affect the automated assignment of cross sections
  - produce wrong answers



example with 1 block but various material regions

User Objects, Postprocessors, AuxVariables and AuxKernels

Integrate a Variable over the Domain

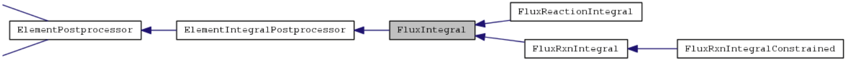
- You have the scaled power density: total_power_density ($p(\vec{r}, t)$) and want to compute total power $P(t)$

$$P(t) = \int_V dVp(r,t)$$

- Simple example of using ElementIntegralVariablePostprocessor

```
[./power]
  type = ElementIntegralVariablePostprocessor
  block = fuel1
  # this is the scaled power density added by the DepletionAction in MAMMOTH (see AuxVariables section)
  variable = total_power_density
[../]
```

Subdomain Integrated Fluxes



- If you want the sum of fluxes integrated over a region (block) in space

$$\text{FluxIntegral} = \sum_{g=g_s}^{g_e} \int_V dV \phi_g$$

  use the FluxIntegral postprocessor.
- Example computes flux integral of the sum of groups 1 and 2 over blocks (regions) 0 and 1 and multiplies it by 1.2.

```
[./flux_integra]
  type = FluxIntegral
  coupled_flux_groups = 'phi_g0 phi_g1'
  block = '0 1'
  scaling_factor = 1.2
[../]
```

Subdomain Integrated Reaction Rates

- If you want the reaction rates integrated over a region in space and over a sum of groups:

$$\text{ReactionRate} = \sum_{g=g_s}^{G} \int_V dV \Sigma_g \phi_g$$

  - FluxReactionIntegral: vector of scalar material properties (Σ_t).
  - FluxRxnIntegral: vector material property ($\nu\Sigma_f$).
- Example computes the integral of the total reaction rate over groups 2 and 3 and block 1.

```
[./flux_reaction]
  type = FluxReactionIntegral
  coupled_flux_groups = 'phi_g1 phi_g2'
  cross_section = 'sigma_total_g1 sigma_total_g2'
  block = '1'
[../]
```

- Example computes the integral of the fission neutron production over groups 2 and 3 and block 1.

```
[./flux_rxn]
  type = FluxRxnIntegral
  coupled_flux_groups = 'phi_g1 phi_g2'
  cross_section = nu_sigma_fission
  initial_group = 2
  block = '1'
[../]
```

Normalization Factor for Eigenvalue Calculations

- Flux obtained in eigenvalue calculations can be arbitrarily scaled
- Scaling factor is obtained by matching total reactor power
- If Depletion block is used the provided reactor power total_reactor_power is correctly scaled
- Example of obtaining scaled power manually (assume two group diffusion):

```
[AuxVariables]
  [./scaled_power]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]
[Postprocessors]
  [./scale_pp]
    type = PowerModulateFactor
    power_pp =  raw_power_pp
    rated_power = 3000e6
    execute_on = 'timestep_begin timestep_end'
    power_modulating = PowerModulator
  [../]
  [./raw_power_pp]
    type = FluxRxnIntegral
    coupled_flux_groups = 'sflux_g0 sflux_g1'
    cross_section = kappa_sigma_fission
    # block needs to be set to wherever kappa_fission exists
    # block = <where kappa fission exists>
    # this is the correct setting for depletion; always true?
    execute_on = 'timestep_begin timestep_end'
  [../]
[]
```

Normalization Factor for Eigenvalue Calculations

```
[Functions]
  # sets power modulating to 1 so integral(power) = rated_power
  [./PowerModulator]
    type = PiecewiseLinear
    x = '0.0 10000'
    y = '1.0 1.0'
  [../]
[]
[AuxKernels]
  [./scaled_power_aux]
    type = VectorReactionRate
    variable = scaled_power
    scalar_flux = 'sflux_g0 sflux_g1'
    # name of the fission heat production cross section
    nusigf = kappa_sigma_fission
    # block needs to be set to wherever kappa_fission exists
    # block = <where kappa fission exists>
    scale_factor = scale_pp
    dummy = raw_power_pp
    # this is the correct setting for depletion; always true?
    execute_on = 'timestep_begin timestep_end'
  [../]
[]
```

- Another example for obtaining scaled fast flux can be found in AuxVariable section
- Scaling in MAMMOTH is tricky because the user can run into cyclic dependencies

Obtaining Reaction Rates and Fluxes Everywhere

- Use UserObjects VariableCartesianCoreMap and FluxCartesianCoreMap
- VariableCartesianCoreMap: Evaluates integrated values of variables on blocks, materials, regions, assemblies, pins, and pin rings/sectors
- FluxCartesianCoreMap: Interacts with the TransportSystem to compute the following reaction rates: fission neutron production, power density, neutron absorbtion, total fluxes, group fluxes
- Can output to dedicated file and/or screen
- Assemblies, pins, and pin rings/sectors can be inferred in two different ways:
  - Define a regular Cartesian grid using input options of UserObject
  - Load variable id values from the mesh file into elemental AuxVariables. (NOTE: variable names are fixed, see manual)

FluxCartesianCoreMap - Example

- Using a cartesian mesh

```
[./flux_map]
 type = FluxCartesianCoreMap
 # this is the name of the transport system (subblock under TransportSystem)
 transport_system = diff
 # the file name used for printing the output
 output_in = flux
 # in this case we use a regular grid
 # always check the grid to make sure that mesh elements are not cut
 regular_grid = true
 grid_coord_x = '0 15 30 45 60 75 90 105 120 135'
 grid_coord_y = '0 15 30 45 60 75 90 105 120 135'
 execute_on = 'initial timestep_end'
[../]
```

- Using elemental variables (see manual for variable names)

```
 [./flux_map]
   type = FluxCartesianCoreMap
   transport_system = saaf
   # selects the quantities to be printed (multiple choices are permitted)
   print = 'assembly pin'
   # determines which cross section is used to evaluate power
   power_map_from = kappa_sigma_f
   execute_on = 'initial timestep_end'
[../]
```

Auxiliary Variables in MAMMOTH

- Auxiliary Variables $\equiv$ Variables that can be readily computed from other quantities without nonlinear solve
- Auxiliary variables are computed by AuxKernels
- AuxKernels can use nonlinear variables, material properties, postprocessors, etc.
- Example: reaction rates, burnup in macroscopic depletion, normalized reactor power, decay heat, boron content, temperatures (in certain models), scaled fast flux etc.

Macroscopic Depletion & Decay Heat Auxiliary Variables

- Macroscopic depletion solves eigenvalue problems at each time step
- Requires scaling of the power to get burnup and decay heat magnitude right
- Depletion block automatically adds:
  - Scaled power density: total_power_density
  - Scaling factor (postprocessor): power_scaling ≡ number to multiply computed flux to get actual reactor flux
  - Burnup: burnup (currently only one unit - select FIMA — MWD/kg — EFPD — J/cm3 (default))
  - **NOTE**: your library must be tabulated in the unit you select here
  - This will soon be generalized
- DecayHeat block automatically adds:
  - Delayed power density: decay_heat_power_density
  - Prompt power density: prompt_power_density
  - Total reactor power density ≡ decay_heat_power_density + prompt_power_density

Macroscopic Depletion & Decay Heat Actions - Example

```
[Depletion]
  # you have to provide the name of the transport system, in this case [./diff]
  transport_system = diff
  family = MONOMIAL
  order = CONSTANT
  # you provide a modulating function for the power (0 <= mod <=1).
  # power_modulator * rated_power = Reactor operating power
  # shutdown mode is entered if this function’s value is zero
  power_modulating_function = PowerModulator
  # rated power in MW: 3000MW thermal for 50,000 fuel pins
  rated_power = 0.06
  # unit that burnup will be computed as, currently only one unit and one fuel type can be selected
  burnup_unit = MWd/kg
   # fuel volume in (mesh unit of length)^3, e.g. cm ∼150 cm^3
  fuel_volume = 150
  # in kg / m^3
  fuel_density = 10421.5
  heavy_metal_isotopes = ’U234 U235 U236 U238’
  weight_percentages = ’0.04 4.11 0.0 95.85’
[]
[DecayHeat]
 fract_power_file = ’path/to/fp.xml’
 decay_heat_blocks = ’1 2 3 4 5 6’
 #  total_power_density and burnup are added by depletion action
 power_density_variable = total_power_density
 burnup_variable = burnup
 order = CONSTANT
 family = MONOMIAL
[]
```

- Limitations: currently only one fuel type and one burnup unit can be selected
- This limitation will be removed soon

Setting Temperature: Constant and Adiabatic Model

- Setting temperature to a constant value (no aux kernel required):

```
[AuxVariables]
  [./Tfuel]
     order = CONSTANT
     family = MONOMIAL
     initial_condition = 566.0
  [../]
[]
```

- Use the adiabatic model (local deposition of energy) for temperature
- $\frac{d(c_p \rho T)}{dt} = p(t) \Rightarrow T(t) = \frac{1}{c_p \rho} \int\limits_0^t d\tau p(\tau)$

```
[AuxKernels]
  [./temperature_aux]
    type = VariableTimeIntegrationAux
    variable = Tfuel
    variable_to_integrate = total_power_density
    # Coeff = 1/(density-Cp), Units [=] K-cm^3/J
    coefficient = 0.732
    execute_on = timestep_end
   # order of the time integration. 2 is default so it’s commented here
   # order = 2
  [../]
[]
```

Setting Boron using a Time-dependent Function

- Boron concentration is often defined using a boron letdown curve
- Boron content is set using a piecewise linear function defined at times $t = 0, 50, 100$ seconds

```
# variable definition
[AuxVariables]
  [./Boron]
    order = CONSTANT
    family = MONOMIAL
    initial_condition = 500.0
  [../]
[]

# function definition
[Functions]
  [./BoronContent]
    type = PiecewiseLinear
    x = '0        50.0    100.0'
    y = '500.0 400.0   100.0 '
  [../]
[]

# AuxKernel definition
[AuxKernels]
   [./SetBoronAux]
     type = FunctionAux
     variable = Boron
     function = BoronContent
   [../]
[]
```

Extracting the Scaled Fast Flux for BISON

- BISON requires the fast flux for estimating radiation damage on
- NOTE: fast flux from eigenvalue calculation is not properly scaled!
- Depletion actions does not set up the properly scaled fast flux so you have to use FluxNormalizationAux and do it yourself

```
# variable definition
[AuxVariables]
  [./scaled_fast_flux]
    # use the same shape function as the primal variable
  [../]
[]

#
[AuxKernels]
  [./scaled_fast_flux_aux]
    type = FluxNormalizationAux
    # this must be the correct name that the transport system assigns to scalar fluxes
    # diffusion: sflux_g<> otherwise flux_moment_g<>_L0_M0
    source_variable = sflux_g0
    variable = scaled_fast_flux
    # this postprocessor is added automatically if Depletion action is used
    normalization = power_scaling
    #  execute this when BISON subapp is executed
    execute_on = timestep_end
  [../]
[]
```

Cyclic Dependencies (Advanced)

- Users could run into the dependency issue:
  - MOOSE provides several *execute_on* options: initial, linear, nonlinear, timestep_begin, timestep_end
  - MOOSE provides (limited) dependency resolution capabilities
  - MOOSE execution order for each execute_on option: PPs (PREAUX)$\rightarrow$ AuxKernels$\rightarrow$ PPs (POSTAUX)
  - PPs are POSTAUX by default and only if AuxKernel depends on them, they become PREAUX
- Illegal (cyclic) dependencies AuxKernel $\rightarrow$ PP $\rightarrow$ AuxKernel on the same *execute_on*
- Problematic dependency: PP1 $\rightarrow$ PP2 $\rightarrow$ AuxKernel, AuxKernel does not know about PP1 and it remains in POSTAUX
- MOOSE does not currently check for this (will be added soon)! User must ensure input is valid
- **NOTE:** The following dependency is correctly resolved: PP $\rightarrow$ AuxKernel $\rightarrow$ P
- Ways to resolve the cyclic dependency:
  - make them have different *execute_on*
  - try making AuxKernel also depend on PP1 in case PP1 $\rightarrow$ PP2 $\rightarrow$ AuxKernel (see FissionSource.C)

Executioners

Executioners - Overview

- List of available executioners or solvers:
  - **Steady**: an executioner in MOOSE framework; for solving steady-state source problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc.
  - **Transient**: an executioner in MOOSE framework; for solving transient problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc with various time integration schemes; Picard iteration for multiphysics can be turned on at each time step.
  - **InversePowerMethod**: an executioner in MOOSE framework; for solving the generalized nonlinear eigenvalue problem; Jacobian-free Krylov method wrapped with the inverse power iterations; Chebyshev acceleration for the power iteration is available.
  - **NonlinearEigen**: an executioner in MOOSE framework; for solving the generalized nonlinear eigenvalue problem; preconditioned Jacobian-free Newton-Krylov (PJFNK) method through PETSc; free inverse power iterations can be activated to ensure the solution converges to the fundamental mode.
  - **Depletion**: a MAMMOTH executioner; for depletion calculations.
- We will discuss **NonlinearEigen**.

Parameters for NonlinearEigen

```
[Executioner]
  [./NonlinearEigen]
   bx_norm                       = (required)

   # convergence control
   source_abs_tol                = 1e-06
   source_rel_tol                = 1e-50

   # advanced
   auto_initialization           = 1
   free_power_iterations         = 4
   k0                            = 1
   pfactor                       = 0.01
   time                          = 0
   output_after_power_iterations = 1

   # normalization
   normal_factor                 =
   normalization                 =
   output_before_normalization = 1
  [../]
[]
```

- Name of the postprocessor for evaluating the $|Bx|$ for the eigenvalue
- Absolute tolerance on residual norm
- Relative tolerance on residual norm after free power iterations
- True to ask the solver to set initial
- The number of free power iterations
- Initial guess of the eigenvalue
- The factor of linear residual norm to be reduced per free power iteration or per nonlinear step
- System time
- True to output solution after free power iterations
- Normalize x to make $|x|$ equal to this factor
- Name of the postprocessor for evaluating $|x|$ for normalization
- True to output a step before normalization

- It typically requires few free power iterations at the beginning to get a better initial guess for converging to the fundamental mode.
- Transport system will automatically add the eigen postprocessor and link it to *bx_norm*.
- Extra parameter can be added for tuning PETSc.

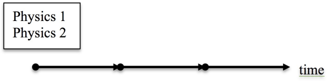
Coupling Physics

www.inl.gov

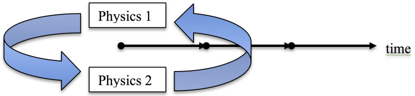
Idaho National Laboratory

Physics Coupling - Definitions

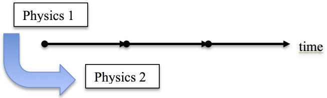
- **Strong** coupling here refers to the implicit solve for all the unknowns without operator splitting - 1 multiphysics matrix.



- **Tight** coupling refers to the split operator for different grouped physics with an external iteration to resolve nonlinearities.



- **Loose** coupling refers to the split operator where a group of physics is lagged in time.



Strong Coupling

www.inl.gov

Idaho National Laboratory

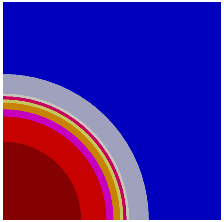
Strong Coupling - Overview

- Currently the physical domain has to be shared and meshing has to match for the implicit solve.
- Currently all physics must be resolved on the same time discretization.
- The Transport Systems will add all neutronics primal variables and kernels automatically.
- Neutronics materials that allow coupling (interpolation of XS) must be defined.
- The rest of the physics (e.g. thermal-fluids, thermo-mechanics, etc.) need to be specified in the same input:
  - Primal variables
  - Primal kernels
  - Materials defining the coefficients for the coupled physics
  - Boundary conditions

Strong Coupling - Exercise

Problem Statement:

- Quarter Fuel Pin with six neutron sub-regions in the fuel pellet
- Temperature obtained in Fuel, Gap, and Clad
- Thermal conduction with heat generated from fission
- Dirichlet Temperature condition specified on the Clad boundary (600K)
- Neutron flux obtained in Fuel, Gap, Clad and Water
- Neutron steady state eigenvalue problem
- Reflective Neutron flux condition specified top, bottom, left and right side boundary



Strong Coupling - Syntax

- Open file exercise_sc_1.i
- Open mesh file to look at blocks and sidesets
- Enter the following:
  - Neutron eigenvalue problem
  - Coarse energy groups - 2
  - B.C. - Reflecting on sidesets '1 2 3'
  - Solver scheme - CFEM-Diffusion
  - Spatial functional family - LAGRANGE
  - Spatial functional order - FIRST

```
[TransportSystems]
  particle =
  equation_type =
  G =
  ReflectingBoundary =

  [./diff]
    scheme =
    family =
    order =
  [../]
[]
```

Strong Coupling - Syntax

- Add the primal variable *TempFuel*
  - first order Lagrange family
  - in all blocks 1-8
  - with initial condition set to 800 K.

```
[Variables]
  [./TempFuel]
    family =
    order =
    block =
    initial_condition =
  [../]
[]
```

- Must add additional kernels:
  - *HeatConduction* operating on *TempFuel* in blocks 1-8
  - *CoupledForce* operating on *TempFuel* in blocks 1-8 coupled to the *ScaledPowerDensity*

```
 [Kernels]
  [./diff]
     type =
     variable =
     block =
  [../]
  [./sourceterm]
     type =
     variable =
     v =
     block =
  [../]
[]
```

Strong Coupling - Additional Syntax

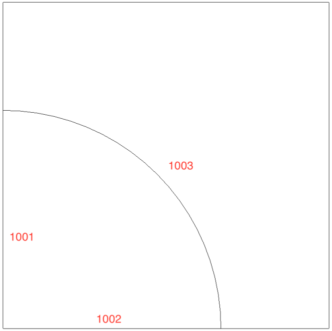
- Material - Javinium
  - is a *HeatConductionMaterial*
  - depends on *TempFuel*
  - has a thermal conductivity of 3.4 W/m/K
  - in blocks 1-6
- Material - Javinium2
  - is a *HeatConductionMaterial*
  - depends on *TempFuel*
  - has a thermal conductivity of 100.0 W/m/K
  - in blocks 7-8
- About Javinium's density
  - constant density (use GenericConstantMaterial)
  - the property name is 'density'
  - with a value of 10900.0 kg/m^3
  - in blocks 7-8

```
[Materials]
  [./Javinium]
     type =
     temp =
     thermal_conductivity =
     block =
  [../]
  [./Javinium2]
     type =
     temp =
     thermal_conductivity =
     block =
  [../]
  [./JaviniumDensity]
     type =
     prop_names =
     prop_values =
     block =
  [../]
[]
```

Strong Coupling - Additional Syntax

- The thermal system requires BCs on *TempFuel*
- place a NeumannBC on sidesets 1001 1002.
- place a DirichletBC on sideset 1003 with a value of 600 K.

```
[BCs]
 active = 'WaterEdgeTempBC
             FuelEdgeTempBC'
 [./FuelEdgeTempBC]
    type =
    variable =
    boundary =
    value =
 [../]
 [./WaterEdgeTempBC]
    type =
    variable =
    boundary =
    value =
 [../]
[]
```



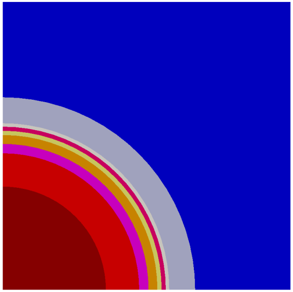
Tight Coupling

Tight Coupling - Overview

- Tight coupling allows different physics systems to be solved for on different domains and different meshes.
- Tight coupling allows for different time discretization to be applied to different physics.
- Implies one master application and many sub applications (slaves).
- Currently, one must prepare one input file for the master and slaves.
- The master
  - does not necessarily have to be the neutronics,
  - contains the execution call of the sub-application via MOOSE's Multi-App system,
  - contains the transfer of information to and from the slave,
  - contains aux variables that store the solution information.
- The Transport Systems will add all neutronics primal variables and kernels automatically.
- Neutronics materials that allow coupling (interpolation of XS) must be defined.

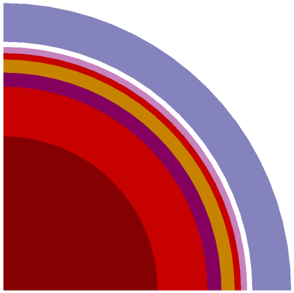
Tight Coupling - Problem Statement Neutronics

- Quarter Fuel Pin with six neutron sub-regions in the fuel pellet
- Neutron flux obtained in Fuel, Gap, Clad and Water
- Neutron steady state eigenvalue problem
- Reflective Neutron flux condition specified top, bottom, left and right side boundary
- Here to leverage the Picard iteration system we leverage the Depletion executioner with two timesteps.



Tight Coupling - Problem Statement Thermal Mechanics

- Temperature obtained in Fuel and Clad
- Fuel is allowed to undergo thermal expansion
- Thermal conduction with heat generated from fission
- Dirichlet Temperature condition specified on the Clad boundary (600K)
- Small heat capacity to minimize impact of the temerpature time derivative



Tight Coupling - Syntax Neutronics File

- Add the primal variable *TempFuel*
  - first order Lagrange family
  - in all blocks 1-8
  - with initial condition set to 900 K.
  - values are obtained from the multiapp transfer

```
[AuxVariables]
   [./TempFuel]
     family =
     order =
     block =
     initial_condition =
   [../]
[]
```

- Add call for multi-app with input file

```
[MultiApps]
  [./sub]
    type = TransientMultiApp
    app_type = MammothApp
    positions = '0.0 0.0 0.0'
    input_files = exercise_tc_1_sub_solution.i
  [../]
[]
```

Tight Coupling - Syntax Neutronics and Fuel Performance File

- Add call for multi-app transfers

```
  [Transfers]
  [./tosub]
    type = MultiAppInterpolationTransfer
    direction = to_multiapp
    multi_app = sub
    source_variable = ScaledPowerDensity
    variable = ScaledPowerDensity
   [../]
   [./fromsub]
    type = MultiAppInterpolationTransfer
    direction = from_multiapp
    multi_app = sub
    source_variable = TempFuel
    variable = TempFuel
   [../]
[]
```

- **Fuel Performance:** Add auxvariables for multi-app transfers

```
[AuxVariables]
  [./ScaldedPowerDensity]
  [../]
[]
```

Detailed Examples

www.inl.gov

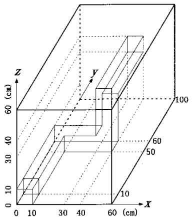
Idaho National Laboratory

Examples - Kobayashi Benchmark

- one-group source problem with void
- Cross sections and source strength:

Region	S $(n \cdot cm^{-3} \cdot s^{-1})$	Σ_t (cm^{-1})	Σ_s (cm^{-1})	
1	1	0.1	0	0.05
2	0	10^{-4}	0	0.5×10^{-4}
3	0	0.1	0	0.05

- Reflective boundary conditions are used at the boundary planes x = 0, y = 0 and z = 0, and vacuum boundary conditions at all outer boundaries



```
[Mesh]
 type = CartesianMesh
 dim = 3

 dx = ’10 20 10 20’
 dy = ’10 40 10 40’
 dz = ’10 20 10 20’

 ix = ’2 4 2 4’
 iy = ’2 8 2 8’
 iz = ’2 4 2 4’

 subdomain_id = ’
   1 3 3 3
   2 3 3 3
   2 2 2 3
   3 3 2 3
```

```
    3 3 3 3
    3 3 3 3
    3 3 2 3
    3 3 3 3

    3 3 3 3
    3 3 3 3
    3 3 2 3
    3 3 2 3

    3 3 3 3
    3 3 3 3
    3 3 3 3
    3 3 3 3’
 uniform_refine = 0
 second_order = false
[]
```

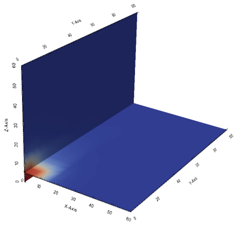
```
[TransportSystems]
  particle = common
  equation_type = steady-state
  G = 1
  VolumetricSourceBlock = ’1’
  VolumetricSource = ’1.0’
  VacuumBoundary = ’right front top’
  ReflectingBoundary = ’left back bottom’

  [./saaf]
    scheme = SAAF-CFEM-SN
    AQtype = Level-Symmetric
    AQorder = 8
    order = FIRST
    hide_angular_flux = true
  [../]
[]
```

Examples - Kobayashi Benchmark - continued

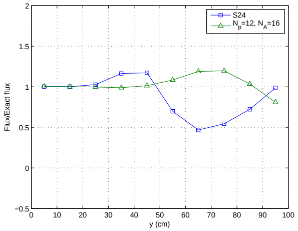
```
[Materials]
  [./region13]
    type = ConstantNeutronicsMaterial
    block = '1 3'
    sigma_t = 0.1
    sigma_s = 0.0
#     sigma_s = 0.05
  [../]
  [./region2]
    type = ConstantNeutronicsMaterial
    block = 2
    sigma_t = 0.0001
    sigma_s = 0.0
#     sigma_s = 5e-5
  [../]
[]

[Postprocessors]
  [./norm]
    type = ElementL2Norm
    variable = flux_moment_g0_L0_M0
  [../]

  [./3A01]
    type = PointValue
    variable = flux_moment_g0_L0_M0
    point = '5 5 5'
  [../]
  ...
[]
```

```
[Executioner]
  type = AMGUpdate
  richardson_max_its = 10
  richardson_abs_tol = 1e-8
  debug = false
  amg_tol = 1e-3
  amg_abs_tol = 1e-9
#   type = Steady
#   petsc_options_iname = '-pc_type -pc_hypre_type
#                          -ksp_gmres_restart'
#   petsc_options_value = 'hypre boomeramg 10'
#   l_max_its = 50
#   nl_rel_tol = 1e-12
[]

[Outputs]
  exodus = true
  print_perf_log = true
  [./csv]
    type = CSV
    file_base = kobayashi_out
    align = true
    precision = 6
    execute_on = timestep_end
  [../]
[]
```

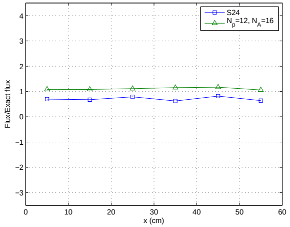
Examples - Kobayashi Benchmark Results



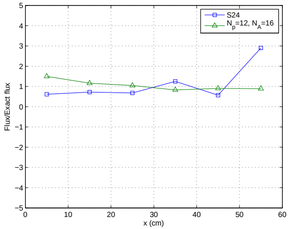
Scalar flux with ParaView



$x = z = 5cm$



$y = 55cm, z = 5cm$



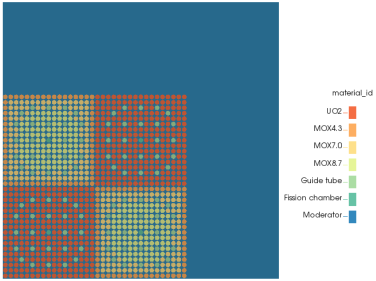
$y = 95cm, z = 35cm$

Examples - C5G7-2D Benchmark

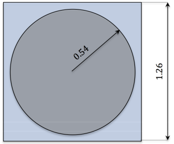
- benchmark on deterministic transport calculations without spatial homogenization.
- two UOX and two MOX assemblies surrounded by moderator.
- each assembly is made up of 17-by-17 grid of pin cells with 264 fuel pins, 24 guide tubes and 1 centering fission chamber.
- seven-group are given for 7 different materials.
- three of seven groups are fast, i.e. no upscattering; four are themal:

Group	Upper Energy
1	20 MeV
2	1 MeV
3	500 keV
4	3 eV
5	0.625 MeV
6	0.1 MeV
7	0.02 MeV

- left and bottom are reflective; right and top are vacuum.

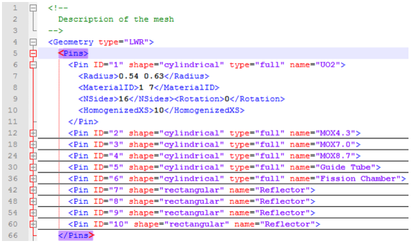


Material assignment

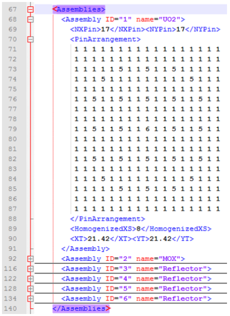


Pin-cell geometry

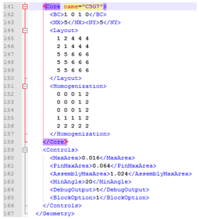
Examples - C5G7-2D Benchmark - Mesh Generation



```
<!--
  Description of the mesh
-->
<Geometry type="LWR">
  <Pins>
    <Pin ID="1" shape="cylindrical" type="full" name="UO2">
      <Radius>0.54 0.63</Radius>
      <MaterialID>1 7</MaterialID>
      <NSides>16</NSides><Rotation>0</Rotation>
      <HomogenizedXS>10</HomogenizedXS>
    </Pin>
    <Pin ID="2" shape="cylindrical" type="full" name="MOX4.3">
    <Pin ID="3" shape="cylindrical" type="full" name="MOX7.0">
    <Pin ID="4" shape="cylindrical" type="full" name="MOX8.7">
    <Pin ID="5" shape="cylindrical" type="full" name="Guide Tube">
    <Pin ID="6" shape="cylindrical" type="full" name="Fission Chamber">
    <Pin ID="7" shape="rectangular" name="Reflector">
    <Pin ID="8" shape="rectangular" name="Reflector">
    <Pin ID="9" shape="rectangular" name="Reflector">
    <Pin ID="10" shape="rectangular" name="Reflector">
  </Pins>
```



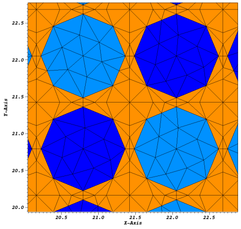
```
  <Assemblies>
    <Assembly ID="1" name="UO2">
      <NXPin>17</NXPin><NYPin>17</NYPin>
      <PinArrangement>
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 5 1 1 5 1 1 5 1 1 1 1 1
        1 1 1 5 1 1 1 1 1 1 1 1 1 5 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 5 1 1 5 1 1 5 1 1 5 1 1 5 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 5 1 1 5 1 1 6 1 1 5 1 1 5 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 5 1 1 5 1 1 5 1 1 5 1 1 5 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 5 1 1 1 1 1 1 1 1 1 5 1 1 1
        1 1 1 1 1 5 1 1 5 1 1 5 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
        1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      </PinArrangement>
      <HomogenizedXS>8</HomogenizedXS>
      <XT>21.42</XT><YT>21.42</YT>
    </Assembly>
    <Assembly ID="2" name="MOX">
    <Assembly ID="3" name="Reflector">
    <Assembly ID="4" name="Reflector">
    <Assembly ID="5" name="Reflector">
    <Assembly ID="6" name="Reflector">
  </Assemblies>
```



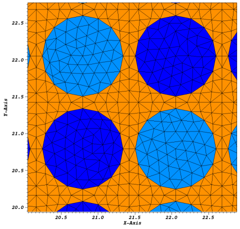
```
  <Core name="C5G7">
    <BC>1 0 1 0</BC>
    <NX>5</NX><NY>5</NY>
    <Layout>
      1 2 4 4 4
      2 1 4 4 4
      5 5 6 6 6
      5 5 6 6 6
      5 5 6 6 6
    </Layout>
    <Homogenization>
      0 0 0 1 2
      0 0 0 1 2
      0 0 0 1 2
      1 1 1 1 2
      2 2 2 2 2
    </Homogenization>
  </Core>
  <Controls>
    <MaxArea>0.016</MaxArea>
    <PinMaxArea>0.064</PinMaxArea>
    <AssemblyMaxArea>1.024</AssemblyMaxArea>
    <MinAngle>20</MinAngle>
    <DebugOutput>t</DebugOutput>
    <BlockOption>1</BlockOption>
  </Controls>
</Geometry>
```

- All sections are contained in *Geometry*.
- The *Pins* section describes all cylindrical and rectangular pin cells.
- The *Assemblies* section describes the pin arrangement of each assembly.
- The *Core* section describes the assembly arrangement, boundary conditions and homogenization level.
- The *Controls* section describes the control parameters for mesh generation.

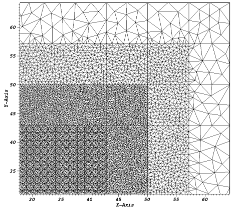
Examples - C5G7-2D Benchmark - Mesh Generation



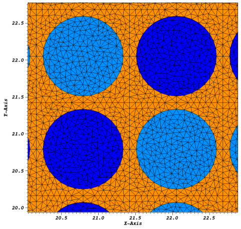
Mesh08 (56.6)



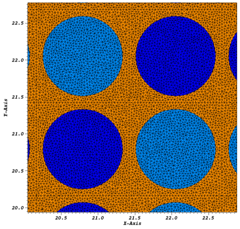
Mesh16 (212.0)



Reflector



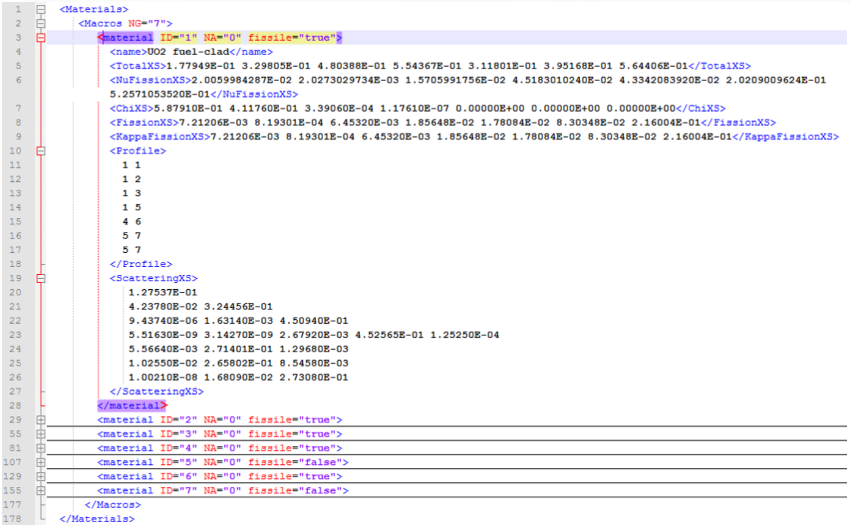
Mesh32 (824.3)



Mesh64 (3370.1)

- The previous XML geometry file is processed by the INSTANT mesh generator by calling Triangle to generate an ExodusII mesh file.
- Fuel rods are approximated with line segments while volumes are preserved.
- We will use Mesh08.

Examples - C5G7-2D Benchmark - Cross Section



```
<Materials>
   <Macros NG="7">
      <material ID="1" NA="0" fissile="true">
        <name>UO2 fuel-clad</name>
        <TotalXS>1.77949E-01 3.29805E-01 4.80388E-01 5.54367E-01 3.11801E-01 3.95168E-01 5.64406E-01</TotalXS>
        <NuFissionXS>2.0059984287E-02 2.0273029734E-03 1.5705991756E-02 4.5183010240E-02 4.3342083920E-02 2.0209009624E-01
        5.2571053520E-01</NuFissionXS>
        <ChiXS>5.87910E-01 4.11760E-01 3.39060E-04 1.17610E-07 0.00000E+00 0.00000E+00 0.00000E+00</ChiXS>
        <FissionXS>7.21206E-03 8.19301E-04 6.45320E-03 1.85648E-02 1.78084E-02 8.30348E-02 2.16004E-01</FissionXS>
        <KappaFissionXS>7.21206E-03 8.19301E-04 6.45320E-03 1.85648E-02 1.78084E-02 8.30348E-02 2.16004E-01</KappaFissionXS>
        <Profile>
          1 1
          1 2
          1 3
          1 5
          4 6
          5 7
          5 7
        </Profile>
        <ScatteringXS>
           1.27537E-01
           4.23780E-02 3.24456E-01
           9.43740E-06 1.63140E-03 4.50940E-01
           5.51630E-09 3.14270E-09 2.67920E-03 4.52565E-01 1.25250E-04
           5.56640E-03 2.71401E-01 1.29680E-03
           1.02550E-02 2.65802E-01 8.54580E-03
           1.00210E-08 1.68090E-02 2.73080E-01
        </ScatteringXS>
      </material>
      <material ID="2" NA="0" fissile="true">
      <material ID="3" NA="0" fissile="true">
      <material ID="4" NA="0" fissile="true">
      <material ID="5" NA="0" fissile="false">
      <material ID="6" NA="0" fissile="true">
      <material ID="7" NA="0" fissile="false">
    </Macros>
</Materials>
```

Examples - C5G7-2D Benchmark

- Set up the input for direct transport solve with SAAF-CFEM-SN.

$$L\left[\Psi\right] = S\left[\Phi\right] + \frac{1}{k} F\left[\Phi\right]$$

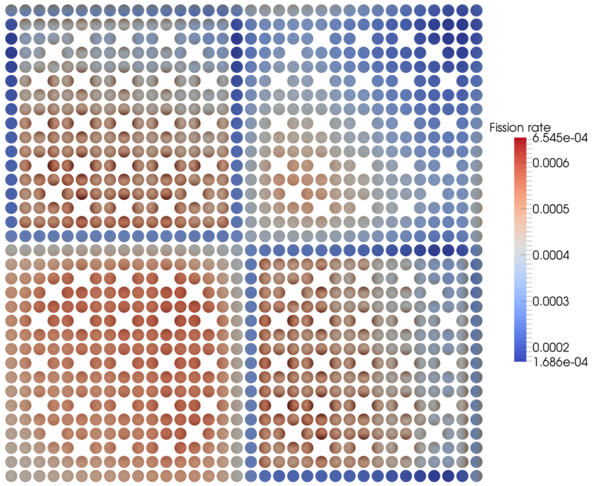
- Convert the direct transport solve for transport update for solving:

$$L\left[\Psi^{(n+1/2)}\right] = S\left[\Phi^{(n)}\right] + \frac{1}{k^n} F\left[\Phi^{(n)}\right]$$

- Set up the low order diffusion system and link it to the transport update.

$$A(\Psi^{(n+1/2)})\left[\Phi^{(n+1)}\right] = S\left[\Phi^{(n+1)}\right] + \frac{1}{k} F\left[\Phi^{(n+1)}\right]$$

Examples - C5G7-2D Benchmark



- Generated with Mesh32, quadratic shape functions, 4 polar directions and 64 azimuthal directions in each octant.
- Axial asymmetry of the fission rate can be observed.
- MOX fuel pins exhibit steeper variation of the fission rate.

More Examples

www.inl.gov

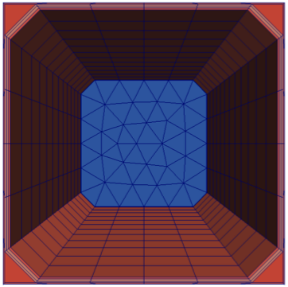
Idaho National Laboratory

TREAT PKE - Overview

- Fully implicit adiabatic heating and PKE
- Uses scalar variables and auxiliary variables
- Uses functions with auxiliary variables and primal variables
- Uses initial conditions and array of initial conditions (special for the PKE set)

TREAT Element - Overview

- 3-D TREAT fuel element with homogenized fuel, gap and clad, but with explicit air channels
- Reflective B.C. on sides and Vacuum top and bottom
- CFEM diffusion solution in 11 energy groups
- Uses the transfer system to generate the I.C. form an eigenvalue calculation
- Transient case via boron dilution
- Includes a fully implicit (strongly coupled) thermal field solution
- thermal properties provided with functions



References

References

1. E.E. Lewis and W.F. Miller, Jr., Computational Methods of Neutron Transport.
2. Yaqi Wang, Mark D. DeHart, Derek R. Gaston, Frederic N. Gleicher, Richard C. Martineau, John W. Peterson, Javier Ortensi, and Sebastian Schunert. **Convergence study of Rattlesnake solutions for the two-dimensional C5G7 MOX benchmark**. Accepted by *Joint International Conference on Mathematics and Computational (M&C), Supercomputing in Nuclear Applications (SNA) and the Monte Carlo (MC) Method, Nashville, TN, USA, April 19-23, 2015.*
3. Sebastian Schunert, Yaqi Wang, Richard C. Martineau. **A New Mathematical Adjoint for the Modified SAAF-SN Equations**. *Annals of Nuclear Energy, v75, p340-352, 2015.*
4. Derek R. Gaston, Cody J. Permann, John W. Peterson, Andrew E. Slaughter, David Andrs, Yaqi Wang, Michael P. Short, Danielle M. Perez, Michael R. Tonks, Javier Ortensi, Richard C. Martineau. **Physics-Based Multiscale Coupling for Full Core Nuclear Reactor Simulation**. *Annals of Nuclear Energy, In Press, Corrected Proof, Available online 5 November 2014.*
5. Frederick N. Gleicher, Javier Ortensi, Benjamin Spencer, Yaqi Wang, Stephen Novascone, Jason Hales, Derek Gaston, Richard Williamson, Richard Martineau. **The Coupling of the Neutron Traonsport Application Rattlesnake to the Nuclear Fuels Performance Application BISON under the MOOSE Framework**. In *Proceedings, International Topical Meeting for Reactor Physics (PHYSOR 2014), Kyoto, Japan, Sept. 28th - Oct. 3rd., 2014.*

References

6. Yaqi Wang, and Frederick N. Gleicher II. **Revisit Boundary Conditions for the Self-Adjoint Angular Flux Formulation**. In *Proceedings, International Topical Meeting for Reactor Physics (PHYSOR 2014), Kyoto, Japan, Sept. 28th - Oct. 3rd., 2014.*
7. Jon Hansen, Jacob Peterson, Jim Morel, Jean Ragusa, Yaqi Wang. **A Least-Squares Transport Equation Compatible with Voids**. *Journal of Computational and Theoretical Transport, Published online, 15 Sep. 2014.*
8. Yaqi Wang, Hongbin Zhang, Richard C. Martineau. **Diffusion Acceleration Schemes for the Self-Adjoint Angular Flux Formulation with a Void Treatment**. *Nuclear Science and Engineering, v176, p201-225, 2014.*
9. Yaqi Wang. **YAKXS - The XML Multigroup Cross Section Library**. *Internal report, 2013.*
10. Frederick N. Gleicher II, Yaqi Wang, Derek Gaston, Richard C. Martineau. **The Method of Manufactured Solutions for RattleSnake, A SN Radiation Transport Solver Inside the MOOSE Framework**. *Transactions of American Nuclear Society, 106(1):372-374, 2012.*

References

11. Yaqi Wang. **NONLINEAR DIFFUSION ACCELERATION FOR THE MULTIGROUP TRANSPORT EQUATION DISCRETIZED WITH SN AND CONTINUOUS FEM WITH RATTLESNAKE**. In *Proceedings, International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2013), Sun Valley, Idaho, USA, May 5-9, 2013.*
12. Hongbin Zhang, Yaqi Wang, Ling Zou, David Andrs, Haihua Zhao, Richard C. Martineau. **Coupling of RELAP-7 with the Three-Dimensional Kinetics Code RattleSnake**. *Transactions of American Nuclear Society, 108(1):863-865, 2013.*
13. Isotopic Analysis of High-Burnup PWR Spent Fuel Samples From the Takahama-3 Reactor (NUREG/CR-6798).
14. Keisuke Kobayashi, Naoki Sugimura, Yasunobu Nagaya, 3-D radiation transport benchmark problems and results for simple geometries with void regions (OECD/NEA, November 2000)
15. Benchmark on Deterministic Transport Calculations Without Spatial Homogenisation, a 2-D/3-D MOX Fuel Assembly Benchmark. (NEA/NSC/DOC(2003)16)
16. Argonne code center: benchmark problem book anl-7416 Supplement 2, Mathematics and Computers (UC-32), June 1977.
17. Frederick Gleicher, Mark DeHart, Javier Ortensi, Yaqi Wang, Anthony Alberti, Evaluation of Multi-Physics Methods to Support Restart of the Transient Test Reactor, INL/CON-15-34177

THE END